分布式存储

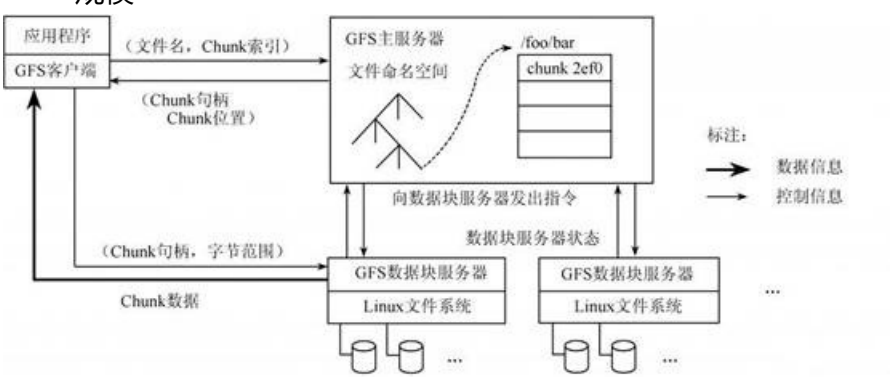
2020年8月17日 21:17

-
- 分布式存储

1. 分布式存储
    a. 问题引入
        i. 互联网上可访问的信息数量
        ii. 海量的数据的有效访问
        iii. 瓶颈：存储技术
    b. 数据存储的体系结构与计算机系统的体系结构密切相关：
        i. 集中式体系结构 -----> 集中式
        ii. 计算机的联网 -----> 客户/服务器
        iii. 分布计算能力 -----> 分布式
    c. 集中式存储技术将数据存储在一台机器或一个节点
    d. 分布式存储
        i. 数据分散的存储
            1) 充分使用网络中的每台机器上的磁盘空间
            2) 并将这些分散的存储资源构成一个虚拟的存储设备
            3) 物理上分散，**逻辑上整体**
        ii. 存储类型
            1) 结构化数据：严格的用户定义的数据类型
                a) 属性
                b) 数据类型
                c) 某种结构组合-如二维表
            2) 非结构化数据
                a) 数据格式不统一
                b) 数据类型多变
                c) 不方便用数据库二维逻辑表来表现
                d) 通常解决方案-分布式文件系统
                    i) GFS
                    ii) HDFS
            3) 半结构化数据
                a) 就是介于完全结构化数据（如关系型数据库、面向对象数据库中的数据）和完全无结构的数据（如声音、图像文件等）之间的数据
                b) 半结构化数据模型具有一定的结构性，但较之传统的关系和面向对象的模型更为灵活
                c) 半结构数据模型完全不基于传统数据库模式的严格概念

d) 不适合用传统的关系型数据库进行存储，适合存储这类数据的数据库被称作“NoSQL”数据库

2. 结构化数据存储
   a. 数据复制：系统维护关系r的几个完全相同的副本（拷贝），各个副本存储在不同的节点上
      i. 全复制：系统中的每个节点都存有关系r的一个拷贝
      ii. 主副本：指定关系r的多个副本中的一个作为主副本，从而简化副本管理
   b. 数据分片：将关系r划分为多个片段r1，r2，…，rn。这些片段中包含足够的信息，使得能够重构原始关系r
      i. 水平分片：将关系r划分为多个子集r1，r2，…，rn。r的每个元组必须至少属于一个片段
         1) 在客户端编程实现路由
         2) 用DRDS、mycat等中间件进行路由
      ii. 垂直分片：将关系r（R）投影到R的属性的多个子集R1，R2，…，Rn。为保证关系r能被重构需要在每个Ri中都包含R的主码属性，或特殊的tuple-id属性
      iii. 混合分片：在水平分片或垂直分片的结果上再进行垂直分片或水平分片
   c. 数据复制与分片
   d. 不复制也不分片
3. GFS（Google File System）
   a. Google 三大论文之一：MapReduce、GFS、BigTale
   b. GFS是实现有史以来最强大通用机群的关键技术之一
   c. 不仅仅是一个文件系统，还包含数据冗余、支持低成本的数据快照，除了提供常规的创建、删除、打开、关闭、读、写文件操作，还提供附加记录的操作
   d. 三类角色
      i. Client（客户端）：是GFS提供给应用程序的访问接口，它是一组专用接口，应用程序直接调用这些库函数，并与该库链接在一起
      ii. Master（主服务器）：是GFS的管理节点，主要存储与数据文件相关的元数据，而不是Chunk（数据块）
      iii. Chunk Server（数据块服务器）：负责具体的存储工作，用来存储Chunk。GFS采用副本的方式实现容错，每一个Chunk有多个存储副本（默认为三个）。 Chunk Server的个数可有有多个，它的数目直接决定了GFS的规模
   e.



4. NOSQL数据库/Non Relational Database(非关系型数据库)
    a. 非关系型，分布式，轻量级，支持水平扩展且一般不保证遵循ACID原则
    b. 兴起原因
        i. 关系数据库已经无法满足Web2.0的需求。主要表现在以下几个方面：
            1) 无法满足海量数据的管理需求
            2) 无法满足数据高并发的需求
            3) 无法满足高可扩展性和高可用性的需求
        ii. “One size fits all”模式很难适用于截然不同的业务场景
            1) 关系模型作为统一的数据模型既被用于数据分析，也被用于在线业务。但这两者一个强调高吞吐，一个强调低延时，已经演化出完全不同的架构。用同一套模型来抽象显然是不合适的
            2) Hadoop就是针对数据分析
            3) MongoDB、Redis等是针对在线业务，两者都抛弃了关系模型
        iii. 关系数据库的关键特性包括完善的事务机制和高效的查询机制。但是，关系数据库引以为傲的两个关键特性，到了Web2.0时代却成了鸡肋，主要表现在以下几个方面：
            1) Web2.0网站系统通常不要求严格的数据库事务
            2) Web2.0并不要求严格的读写实时性
            3) Web2.0通常不包含大量复杂的SQL查询（去结构化，存储空间换取更好的查询性能）
    c. 关系型vs非关系型
        i. 关系数据库优势：以完善的关系代数理论作为基础，有严格的标准，支持事务ACID四性，借助索引机制可以实现高效的查询，技术成熟，有专业公司的技术支持
        ii. 劣势：可扩展性较差，无法较好支持海量数据存储，数据模型过于死板、无法较好支持Web2.0应用，事务机制影响了系统的整体性能等
        iii. NoSQL数据库优势：可以支持超大规模数据存储，灵活的数据模型可以很好地支持Web2.0应用，具有强大的横向扩展能力等
        iv. 劣势：缺乏数学理论基础，复杂查询性能不高，大都不能实现事务强一致性，很难实现数据完整性，技术尚不成熟，缺乏专业团队的技术支持，维护较困难等
    d. 应用场景
        i. 关系数据库应用场景：电信、银行等领域的关键业务系统，需要保证强事务一致性
        ii. NoSQL数据库应用场景：互联网企业、传统企业的非关键业务（比如数据分析）
        iii. 采用混合架构
            1) 案例：亚马逊公司使用不同类型的数据库来支撑它的电子商务应用
            2) 对于“购物篮”这种临时性数据，采用键值存储会更加高效
            3) 当前的产品和订单信息则适合存放在关系数据库中

4）大量的历史订单信息则适合保存在类似MongoDB的文档数据库中

e. CAP
   i. C（Consistency）：一致性，是指任何一个读操作总是能够读到之前完成的写操作的结果，也就是在分布式环境中，多点的数据是一致的，或者说，所有节点在同一时间具有相同的数据
   ii. A（Availability）：可用性，是指快速获取数据，可以在确定的时间内返回操作结果，保证每个请求不管成功或者失败都有响应
   iii. P（Tolerance of Network Partition）：分区容忍性，是指当出现网络分区的情况时（即系统中的一部分节点无法和其他节点进行通信），分离的系统也能够正常运行，也就是说，系统中任意信息的丢失或失败不会影响系统的继续运作
   iv. CA：MySQL, SQL Server, PostgreSQL等
   v. AP：Dynamo, Cassandra, Voldemort, CouchDB, Riak
   vi. CP：Neo4J, Bigtable, MongoDB, HBase, Hypertable, Redis

f. BASE
   i. 基本可用（Basically Availble）：基本可用，是指一个分布式系统的一部分发生问题变得不可用时，其他部分仍然可以正常使用，也就是允许分区失败的情形出现
   ii. 软状态（Soft-state）：“软状态（soft-state）”是与“硬状态（hard-state）”相对应的一种提法。数据库保存的数据是“硬状态”时，可以保证数据一致性，即保证数据一直是正确的。“软状态”是指状态可以有一段时间不同步，具有一定的滞后性
   iii. 最终一致性（Eventually consistent）：允许后续的访问操作可以暂时读不到更新后的数据，但是经过一段时间之后，必须最终读到更新后的数据。最常见的最终一致性系统是DNS（域名系统）

5. NOSQL逻辑类型

a. 键值数据库：键值对

相关产品	Redis、Riak、SimpleDB、Chordless、Scalaris、Memcached
数据模型	键/值对 键是一个字符串对象 值可以是任意类型的数据，比如整型、字符型、数组、列表、集合等
典型应用	涉及频繁读写、拥有简单数据模型的应用 内容缓存，比如会话、配置文件、参数、购物车等 存储配置和用户数据信息的移动应用
优点	扩展性好，灵活性好，大量写操作时性能高
缺点	无法存储结构化信息，条件查询效率较低
不适用情形	不是通过键而是通过值来查：键值数据库根本没有通过值查询的途径 需要存储数据之间的关系：在键值数据库中，不能通过两个或两个以上的键来关联数据 需要事务的支持：在一些键值数据库中，产生故障时，不可以回滚

使用者	百度云数据库 （Redis）、GitHub（Riak）、BestBuy（Riak）、Twitter（Redis和Memcached）、StackOverFlow（Redis）、Instagram （Redis）、Youtube（Memcached）、Wikipedia（Memcached）

b. 列族数据库：一些列的聚族

相关产品	BigTable、HBase、Cassandra、HadoopDB、GreenPlum、PNUTS
数据模型	列族
典型应用	分布式数据存储与管理 数据在地理上分布于多个数据中心的应用程序 可以容忍副本中存在短期不一致情况的应用程序 拥有动态字段的应用程序 拥有潜在大量数据的应用程序，大到几百TB的数据
优点	查找速度快，可扩展性强，容易进行分布式扩展，复杂性低
缺点	功能较少，大都不支持强事务一致性
不适用情形	需要ACID事务支持的情形，Cassandra等产品就不适用
使用者	Ebay（Cassandra）、Instagram（Cassandra）、NASA（Cassandra）、Twitter（Cassandra and HBase）、Facebook（HBase）、Yahoo!（HBase）

c. 文档数据库：文件名对应文本数据（一般是JSON文本）

相关产品	MongoDB、CouchDB、Terrastore、ThruDB、RavenDB、SisoDB、RaptorDB、CloudKit、Perservere、Jackrabbit
数据模型	键/值 值（value）是版本化的文档
典型应用	存储、索引并管理面向文档的数据或者类似的半结构化数据 比如，用于后台具有大量读写操作的网站、使用JSON数据结构的应用、使用嵌套结构等非规范化数据的应用程序
优点	性能好（高并发），灵活性高，复杂性低，数据结构灵活 提供嵌入式文档功能，将经常查询的数据存储在同一个文档中 既可以根据键来构建索引，也可以根据内容构建索引
缺点	缺乏统一的查询语法
不适用情形	在不同的文档上添加事务。文档数据库并不支持文档间的事务，如果对这方面有需求则不应该选用这个解决方案
使用者	百度云数据库（MongoDB）、SAP （MongoDB）、Codecademy（MongoDB）、Foursquare （MongoDB）、NBC News （RavenDB）

d. 图形数据库：存储结点和边关系

相关产品	Neo4J、OrientDB、InfoGrid、Infinite Graph、GraphDB
数据模型	图结构
典型应用	专门用于处理具有高度相互关联关系的数据，比较适合于社交网络、模式识别、依赖分析、推荐系统以及路径寻找等问题

优点	灵活性高，支持复杂的图形算法，可用于构建复杂的关系图谱
缺点	复杂性高，只能支持一定的数据规模
使用者	Adobe（Neo4J）、Cisco（Neo4J）、T-Mobile（Neo4J）

6. redis入门
    a. www.redis.io
    b. redis持久化：Redis是一个支持持久化的内存数据库，也就是说Redis需要经常将内存中的数据同步到磁盘来保证持久化。Redis支持两种持久化方式
        i. Snapshotting(快照)是默认方式
            1) 编辑redis.conf文件
            2) 例：save 900 10：900秒超过10个key被修改
        ii. Append-Only File(AOF)
    c. 集群环境的session管理
        i. 在tomcat的lib目录中导入
            tomcat-redis-session-manager-1.2-tomcat-6.jar
            jedis-2.0.0.jar
            commons-pool-1.6
        ii. 在tomcat的conf目录中修改context.xml配置文件
            <Valve
            className="com.radiadesign.catalina.session.RedisSessionHandlerValve" />
            <Manager
            className="com.radiadesign.catalina.session.RedisSessionManager" host="192.168.1.105" port="6379"
            maxInactiveInterval="60"/>
    d. 应用场景
        i. String：缓存，计数、共享session等
        ii. Hash：用户信息、购物车、最终购买商品等
        iii. List:消息、文章列表
        iv. Set：标签（给用户添加标签，给标签添加用户）
        v. Zset：排行榜